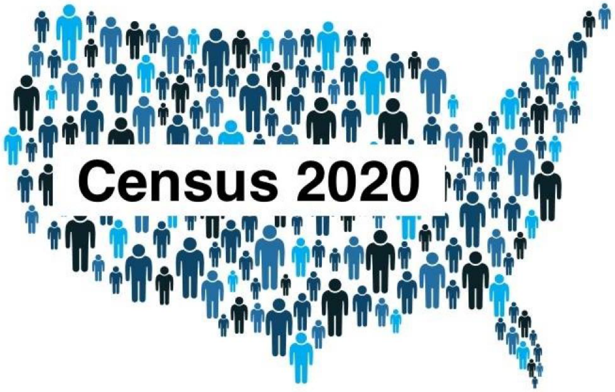
CSV Format/Real-world Data



CS111 Computer Programming

Department of Computer Science
Wellesley College

Recap: File Formats so far

One way to provide input for our programs is through files that store data.
So far we have seen how to work with two file formats: TXT files and JSON files.
In both cases, we have to create first a fileObject that refers to a file that is open for either reading or writing (e.g., `fileObjR` and `fileObjW`).

```
with open(filePath, 'r') as fileObjR:
    # do reading/loading operation

with open(filePath, 'w') as fileObjW:
    # do writing/dumping operation
```

Operation	Python syntax

Reading text from a file	`fileObjR.read()`
	`fileObjR.readline()`
	`fileObjR.readlines()`

Writing text into a file	`fileObjW.write(aStr)`

Operations for working with **TXT** files

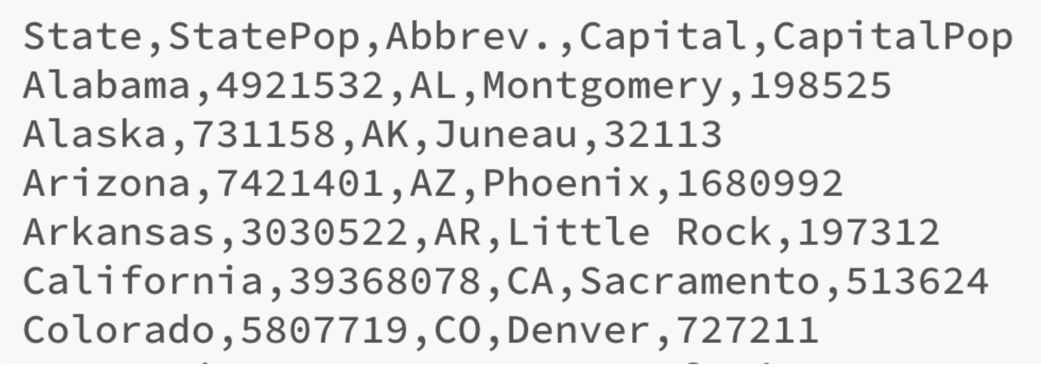
Operation	Python syntax
Loading a JSON object from a file	`json.load(fileObjR)`
Dumping a JSON object into a file	`json.dump(obj, fileObjW)`

Operations for working with **JSON** files

The CSV Format

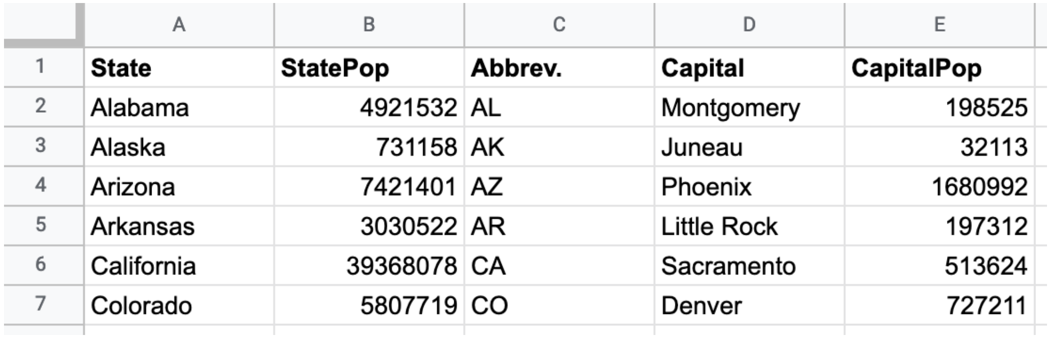
(CSV = Comma Separated Values)

Concepts in this slide: Introducing a new file format for tabular data.



```
State,StatePop,Abbrev.,Capital,CapitalPop
Alabama,4921532,AL,Montgomery,198525
Alaska,731158,AK,Juneau,32113
Arizona,7421401,AZ,Phoenix,1680992
Arkansas,3030522,AR,Little Rock,197312
California,39368078,CA,Sacramento,513624
Colorado,5807719,CO,Denver,727211
```

Partial screenshot of the `us-states-more.csv` file, viewed with a text editor.



	A	B	C	D	E
1	**State**	**StatePop**	**Abbrev.**	**Capital**	**CapitalPop**
2	Alabama	4921532	AL	Montgomery	198525
3	Alaska	731158	AK	Juneau	32113
4	Arizona	7421401	AZ	Phoenix	1680992
5	Arkansas	3030522	AR	Little Rock	197312
6	California	39368078	CA	Sacramento	513624
7	Colorado	5807719	CO	Denver	727211

Partial screenshot of the `us-states-more.csv` file, viewed with the Google Spreadsheet editor.

CSV files are one of the most common formats to share data, since they can be displayed as a table in spreadsheet applications (Microsoft Excel, Google Spreadsheet, etc.).

Reading tuples from CSV files

For simple CSV files, we can write our own function to read its content.

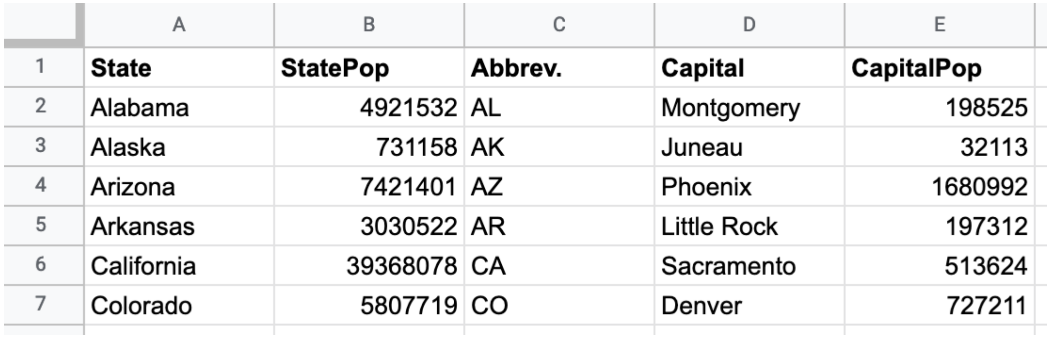
```
def tuplesFromFile(filename):
    '''Read each line from opened file,
    strip white space,
    split at commas,
    convert as tuple and
    return a list of tuples.
    '''
    with open(filename, 'r') as inputFile:
        theTuples = [tuple(line.strip().split(','))
                              for line in inputFile]
    return theTuples
```

To notice:

We are using a list comprehension to read the content of the files into a list of tuples. This statement replaces this code:

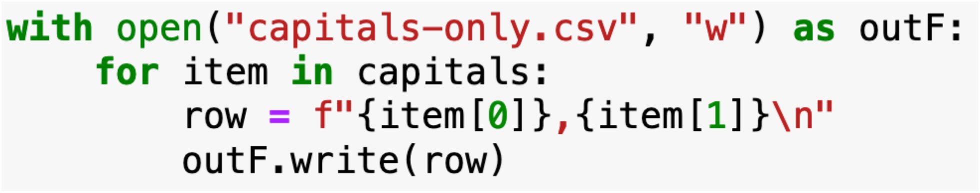
```
theTuples = []
for line in inputFile:
    theTuples.append(tuple(line.strip().split(',')))
```

What happens when our data has commas?

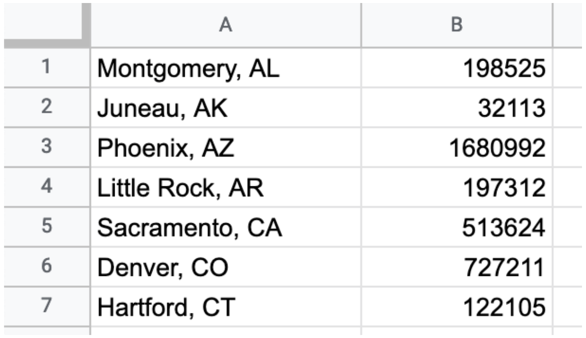


	A	B	C	D	E
1	**State**	**StatePop**	**Abbrev.**	**Capital**	**CapitalPop**
2	Alabama	4921532	AL	Montgomery	198525
3	Alaska	731158	AK	Juneau	32113
4	Arizona	7421401	AZ	Phoenix	1680992
5	Arkansas	3030522	AR	Little Rock	197312
6	California	39368078	CA	Sacramento	513624
7	Colorado	5807719	CO	Denver	727211

Partial screenshot of the `us-states-more.csv` file, viewed with the Google Spreadsheet editor.

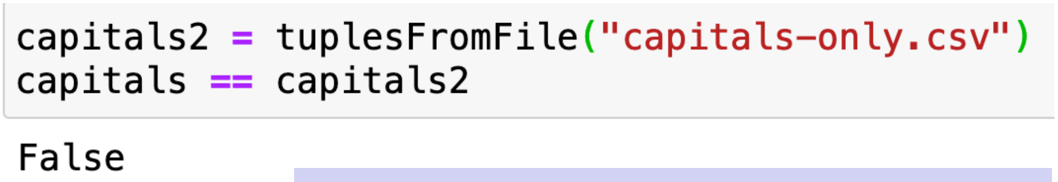


```
with open("capitals-only.csv", "w") as outF:
    for item in capitals:
        row = f"{item[0]},{item[1]}\n"
        outF.write(row)
```



	A	B
1	Montgomery, AL	198525
2	Juneau, AK	32113
3	Phoenix, AZ	1680992
4	Little Rock, AR	197312
5	Sacramento, CA	513624
6	Denver, CO	727211
7	Hartford, CT	122105

Partial screenshot of the `capitals-only.csv` file, viewed with the Google Spreadsheet editor.



```
capitals2 = tuplesFromFile("capitals-only.csv")
capitals == capitals2
```

```
False
```

Check the Notebook

It's easy to create the file about capitals from the state data, but when we read it back using the function **`tuplesFromFile`**, the result has tuples of three values, not two, as we desire.

The `csv` module

The csv module has four functions that create special objects to read/write CSV files.

`csv.reader`	creates an object that reads the content of CSV file as a list of lists
`csv.writer`	creates an object that writes a list of lists into a CSV file
`csv.DictReader`	**creates an object that reads the content of CSV file as a list of dictionaries**
`csv.DictWriter`	**creates an object that writes a list of dictionaries into a CSV file**

Important Note

In CS111, we will only be covering DictReader and DictWriter, since they help us work with dictionaries.

csv.DictReader [1]

Differently from reading/loading TXT and JSON files, reading a CSV file as a dictionary is a two step process:

1. Create a DictReader object that is tied to the file object open for reading
2. Read and convert each line from the text file as a dict object

```
with open('countries.csv', 'r') as inputFile:
    dctReader = csv.DictReader(inputFile)
    rows = [row for row in dctReader] # read line by line
    print(inputFile)
    print(dctReader)
    print(rows)
```

The file object

```
<_io.TextIOWrapper name= ' countries.csv' mode='r' encoding='UTF-8'>
<csv.DictReader object at 0x7f84901a4c10>
```

The DictReader object

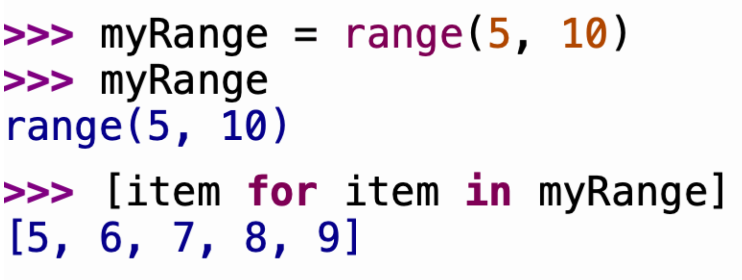
```
[{'country': 'Canada', 'capital': 'Ottawa'}, {'country': 'Mexico', 
'capital': 'Mexico City'}, {'country': 'South Korea', 'capital': 'Seoul'}, 
{'country': 'Ukraine', 'capital': 'Kiev'}]
```

csv.DictReader [2]

`csv.DictReader` creates an iterator object that reads lines into dictionaries only when we “force” it to do the work through iteration.

```
dctReader = csv.DictReader(inputFile)
rows = [row for row in dctReader]
```

This is very similar to how the **`range`** object behaves:



```
>>> myRange = range(5, 10)
>>> myRange
range(5, 10)
>>> [item for item in myRange]
[5, 6, 7, 8, 9]
```

csv.DictWriter

Writing a dictionary into a CSV file involves the following steps:

1. Create a DictWriter object tied to a file open for writing
2. Write the header of the file, which contains the names of the columns
3. Write all dictionaries as rows in the files

```
oscarMovies = [{'title': 'CODA', 'year': 2022},
               {'title': 'Nomadland', 'year': 2021},
               {'title': 'Parasite', 'year': 2020}]

columns = oscarMovies[0].keys() # get the names of the keys

with open('oscarWinners.csv', 'w', newline='') as outFile:
    dctWriter = csv.DictWriter(outFile, fieldnames=columns)
    dctWriter.writeheader() # no need for argument
    dctWriter.writerows(oscarMovies)
```

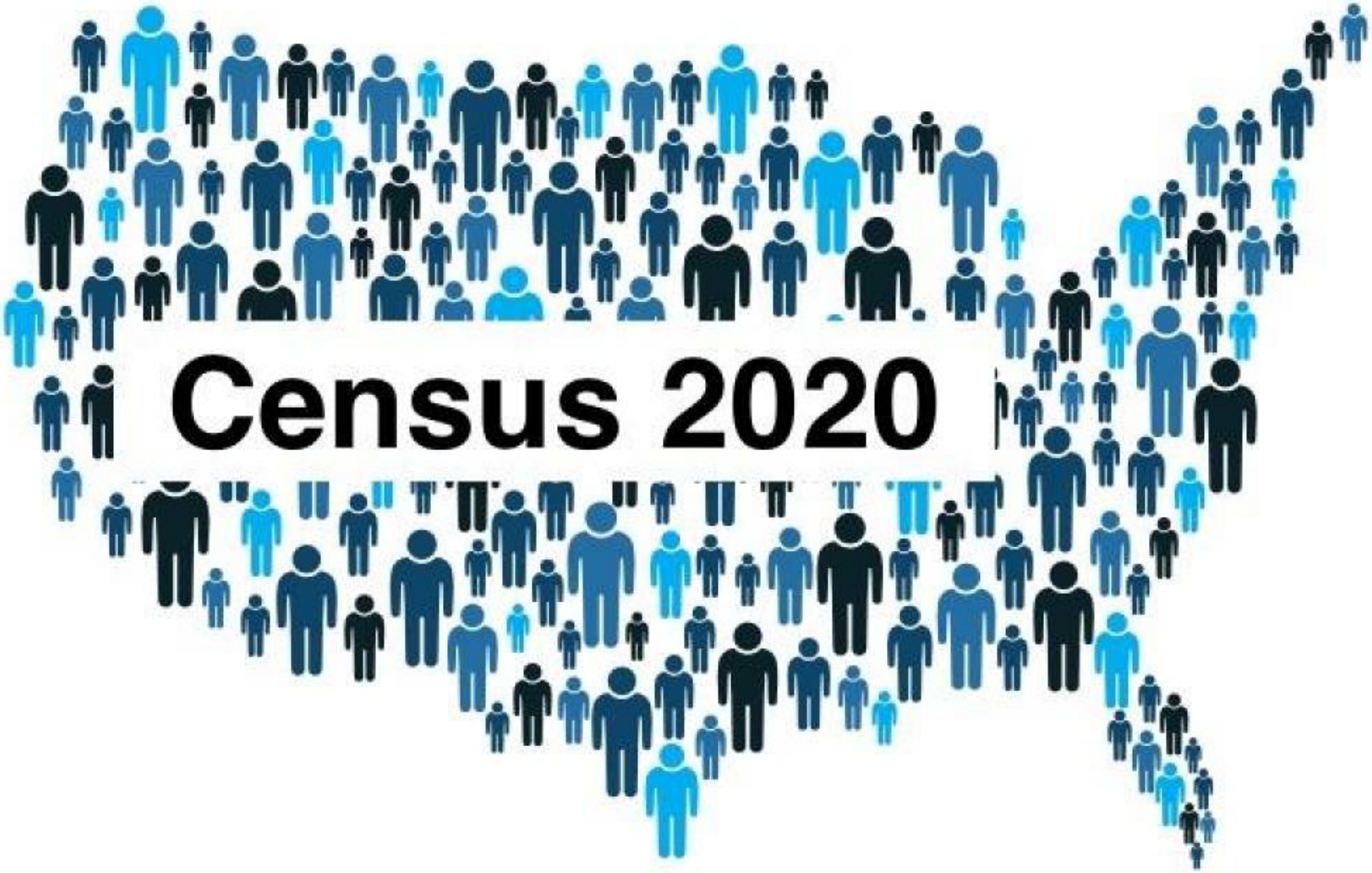
Additional Parameters

Notice that we have added a third parameter to the `open` function: `newline=''`

This is needed to deal with the different way that Windows machines deal with newlines.

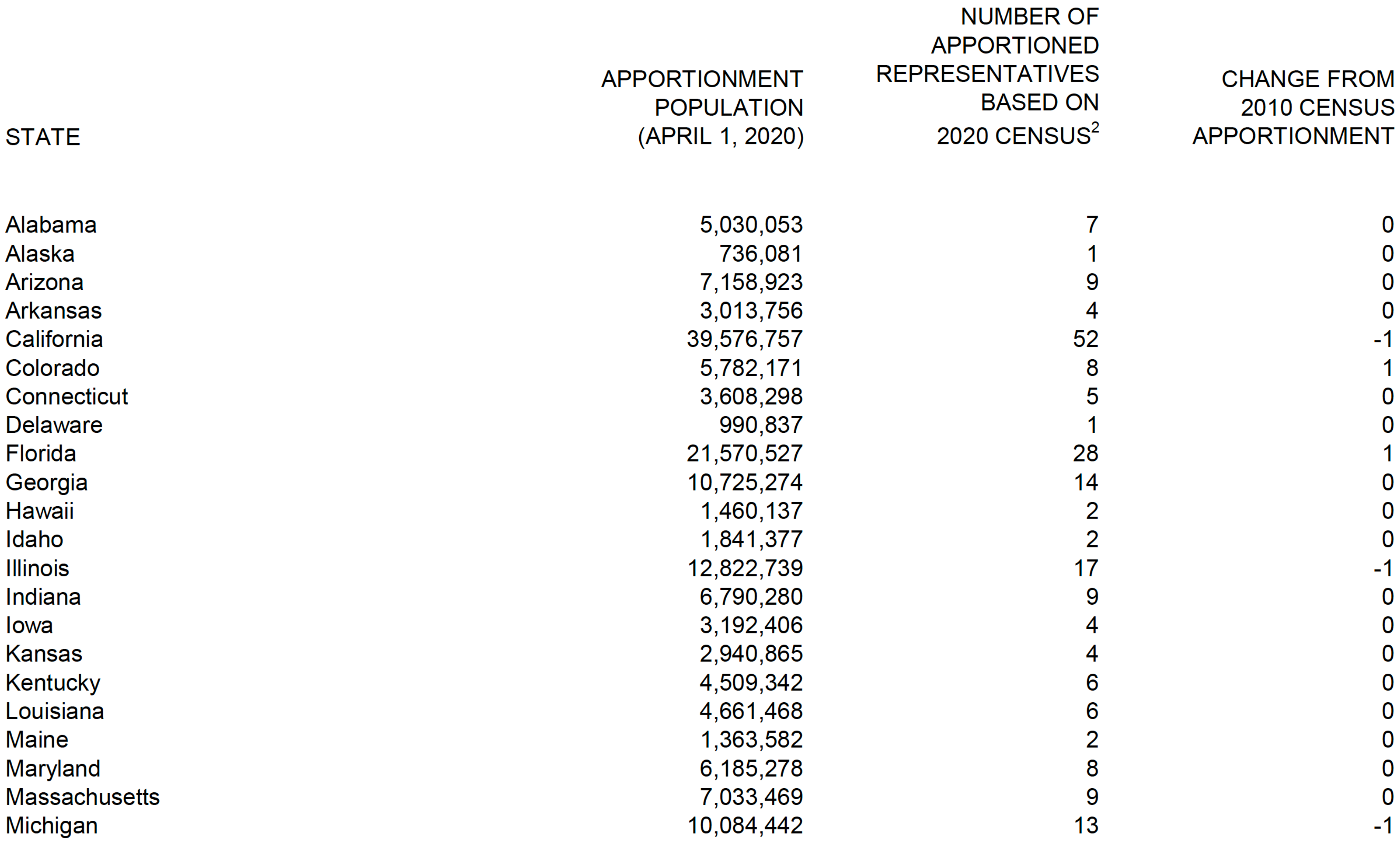
More examples

Check the notebook for examples to understand what **`writeheader`**, **`writerows`**, and one method not shown here, **`writerow`**, do.



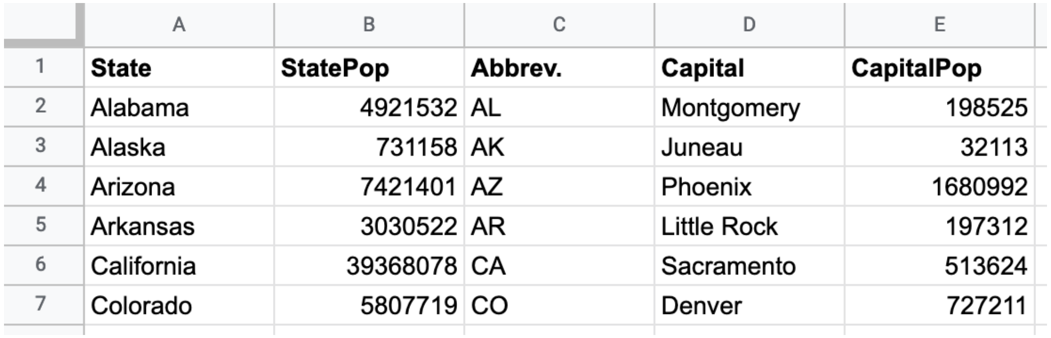
Census 2020

Representation in Congress is based on population. More people, more seats.



STATE	APPORTIONMENT POPULATION (APRIL 1, 2020)	NUMBER OF APPORTIONED REPRESENTATIVES BASED ON 2020 CENSUS[2]	CHANGE FROM 2010 CENSUS APPORTIONMENT
Alabama	5,030,053	7	0
Alaska	736,081	1	0
Arizona	7,158,923	9	0
Arkansas	3,013,756	4	0
California	39,576,757	52	-1
Colorado	5,782,171	8	1
Connecticut	3,608,298	5	0
Delaware	990,837	1	0
Florida	21,570,527	28	1
Georgia	10,725,274	14	0
Hawaii	1,460,137	2	0
Idaho	1,841,377	2	0
Illinois	12,822,739	17	-1
Indiana	6,790,280	9	0
Iowa	3,192,406	4	0
Kansas	2,940,865	4	0
Kentucky	4,509,342	6	0
Louisiana	4,661,468	6	0
Maine	1,363,582	2	0
Maryland	6,185,278	8	0
Massachusetts	7,033,469	9	0
Michigan	10,084,442	13	-1

US States and Capitals: Doing more with our data



	A	B	C	D	E
1	**State**	**StatePop**	**Abbrev.**	**Capital**	**CapitalPop**
2	Alabama	4921532	AL	Montgomery	198525
3	Alaska	731158	AK	Juneau	32113
4	Arizona	7421401	AZ	Phoenix	1680992
5	Arkansas	3030522	AR	Little Rock	197312
6	California	39368078	CA	Sacramento	513624
7	Colorado	5807719	CO	Denver	727211

Partial screenshot of the `us-states-more.csv` file, viewed with the Google Spreadsheet editor.

Some questions to answer with our data:

- Which are the **most** populated US states? **Rank** the data in that order.
- Which are the **least** populated US states? **Rank** the data in that order.
- Which US state capitals are the **most** populated? **Rank** the data in that order.
- Which US state capitals are the **least** populated? **Rank** the data in that order.
- What percentage of each US state's population lives in the state capital? **Rank** the data by that percentage from the **largest** to the **smallest**.

Can dictionaries be sorted? Explain the outputs!

```
In [31]: fruitColors = {"banana": "yellow", "kiwi":
"green", "grapes": "purple", "apple": "red",
"lemon": "yellow", "pomegranate": "red"}
In [32]: sorted(fruitColors)
Out[32]: ['apple', 'banana', 'grapes', 'kiwi',
'lemon', 'pomegranate']
In [33]: sorted(fruitColors.keys())
Out[33]: ['apple', 'banana', 'grapes', 'kiwi',
'lemon', 'pomegranate']
In [34]: sorted(fruitColors.values())
Out[34]: ['green', 'purple', 'red', 'red', 'yellow',
'yellow']
In [35]: sorted(fruitColors.items())
Out[35]: [('apple', 'red'), ('banana', 'yellow'),
('grapes', 'purple'), ('kiwi', 'green'), ('lemon',
'yellow'), ('pomegranate', 'red')]
```

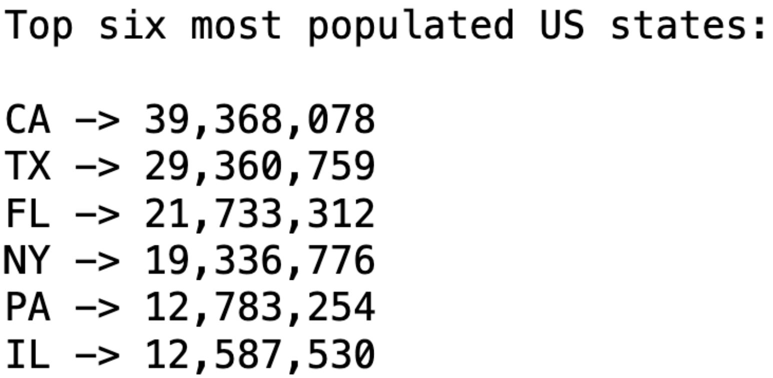
Sort a list of dictionaries

```
In [36]: peopleDctList = [{'name':'Mary Beth Johnson', 'age': 18},
                           {'name':'Ed Smith', 'age': 17},
                           {'name':'Janet Doe', 'age': 25},
                           {'name':'Bob Miller', 'age': 31}]
In [37]: sorted(peopleDctList)
Traceback (most recent call last):
  File "<pyshell>", line 1, in <module>
TypeError: '<' not supported between instances of 'dict' and 'dict'

def byAge(personDct):
    return personDct['age']

In [38]: sorted(peopleDctList, key=byAge, reverse=True)
Out[38]: [{'name': 'Bob Miller', 'age': 31},
          {'name': 'Janet Doe', 'age': 25},
          {'name': 'Mary Beth Johnson', 'age': 18},
          {'name': 'Ed Smith', 'age': 17}]
```

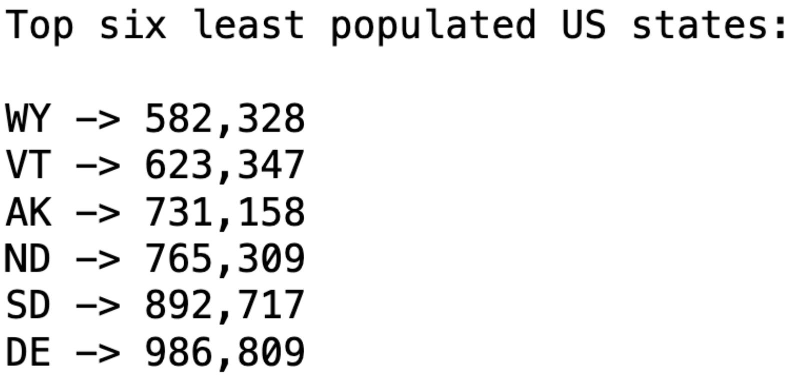
Questions 1 & 2: Sort by US state population

How to implement the solution with Python code:

1. Read the content of the CSV file `us-states-more.csv` using **`csv.DictReader`**, which returns a list of dictionaries.
2. Create a helper function byStatePop, which, given a dictionary with state data (one row from our file), returns the appropriate value. Remember that all values in the dictionary are strings, because they come from the CSV file.
3. Apply the **`sorted`** function to the list of dictionaries of state data, using the **`key`** parameter with the function byStatePop.
4. Look at the results, in which way are they sorted?
5. Include the function parameter **`reverse`** to change the order of sorting.
6. Use f-string formatting to print out top six results as shown below.



```
Top six most populated US states:

CA -> 39,368,078
TX -> 29,360,759
FL -> 21,733,312
NY -> 19,336,776
PA -> 12,783,254
IL -> 12,587,530
```

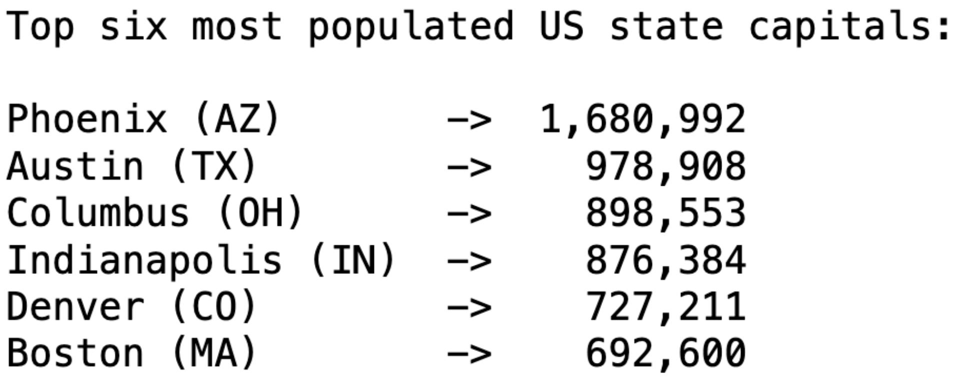


```
Top six least populated US states:

WY -> 582,328
VT -> 623,347
AK -> 731,158
ND -> 765,309
SD -> 892,717
DE -> 986,809
```

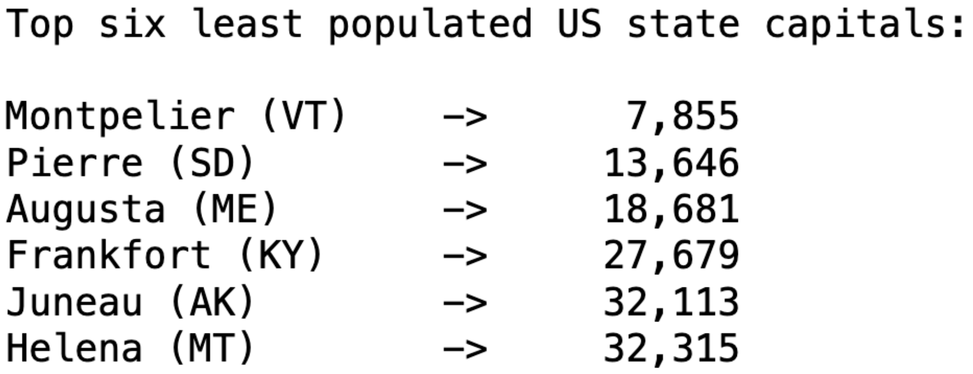
Questions 3 &4: Sort by capital population

How to implement the solution with Python code:

Follow the steps from the previous slide, but create appropriate functions to use with the parameter key for sorted. Try to come as close as possible to these outputs, but don't worry if you cannot. These outputs use some special f-string features for formatting.



```
Top six most populated US state capitals:

Phoenix (AZ)       ->  1,680,992
Austin (TX)        ->    978,908
Columbus (OH)      ->    898,553
Indianapolis (IN)  ->    876,384
Denver (CO)        ->    727,211
Boston (MA)        ->    692,600
```

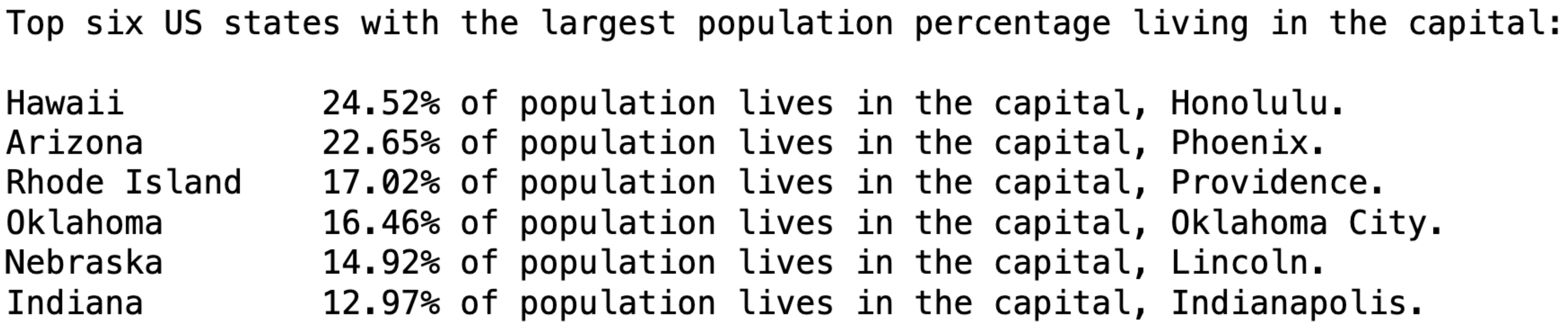


```
Top six least populated US state capitals:

Montpelier (VT)    ->      7,855
Pierre (SD)        ->     13,646
Augusta (ME)       ->     18,681
Frankfort (KY)     ->     27,679
Juneau (AK)        ->     32,113
Helena (MT)        ->     32,315
```

Questions 5: Sort by percentage

How to implement the solution with Python code:

This will be similar to the two previous slides, by you'll have to create a helper function, **`byPercentage`**, which can calculate the percentage of people living in the capital of the state. This function will be used by the **`sorted`** function, as well as by the f-string. Try to come close to this output, but do not worry if you cannot achieve it yet.



```
Top six US states with the largest population percentage living in the capital:

Hawaii          24.52% of population lives in the capital, Honolulu.
Arizona         22.65% of population lives in the capital, Phoenix.
Rhode Island    17.02% of population lives in the capital, Providence.
Oklahoma        16.46% of population lives in the capital, Oklahoma City.
Nebraska        14.92% of population lives in the capital, Lincoln.
Indiana         12.97% of population lives in the capital, Indianapolis.
```

Test your knowledge

1. What do the acronyms JSON and CSV stand for?
2. In what ways do these two formats differ from one another?
3. Which format allows programmers more flexibility in transferring data? Why?
4. What do the two functions `dump` and `load` of the **`json`** module do?
5. What do the two functions `csv.DictReader` and `csv.DictWriter` do?
6. What does the method `writeheader` do?
7. What do we need to do in order to sort a list of dictionaries? Why is that?
8. What are some other questions that you could answer with the Census data. Can you write the Python code to answer them? Try it out and let us know what you did. We might add that in our material for future semesters.